Investigation of RSA and DSA Cryptography Systems

Emmet Rice

Queen's University Belfast

Introduction

In the Information age, with the ubiquitous influence of the internet in everyday communications, and data a prime resource; cryptology is paramount. Cryptography is utilized to encrypt and decrypt these communications, in order to secure the potentially vital information enclosed from malicious adversaries. The "Rivest-Shamir-Adleman" (RSA) algorithm was the first wide spread, and still in use today, asymmetric encryption algorithm. This means that the "secret-key" required to decipher the code is not required to be sent across a potentially insecure channel, reducing the potential risk. This also led to the idea of "Digital Signatures" in order to validate messages, and ultimately the modern verification method via the Digital Signature Algorithm (DSA). MATLAB was chosen for the numerical calculations for this investigation.

RSA Encryption

RSA: Overview

As an asymmetric encryption algorithm, RSA was developed upon the principle that the encryption and decryption operations are the inverse of each other using two different keys, known as the public and private key. As the naming suggests, the public key is shared, and the private key is kept secret. The Public key is used for the encryption operation, and consequently the private key for decryption. This enables anyone to send the receiver, which the public key belongs to, a correspondingly encoded message; while only the receiver may decipher it without involving sharing the secret key over an insecure channel.
The generation of these keys relies on the properties of what are known in cryptography as a "Trapdoor Functions" or "One-way Functions". These functions are trivially easy to compute in one direction, but the inverse direction requires significant computational effort without the knowledge of special information known as the "trapdoor". The significance of this to RSA is readily apparent, and the corresponding RSA trapdoor function is Prime Factorisation; where it is computationally intractable to factorise large prime numbers. Thus, two large, distinct and typically randomly chosen prime factors are used in the RSA key generation process.

The key generation process also utilises the Bézout identity, and the Extended Euclidean Algorithim (EEA) to calculate the Bézout coefficients and the greatest common divisor (gcd) between two integer numbers (a,b). This algorithm is implemented in the EEA.m MATLAB file.

Bézout Identity

$$am + bn = gcd(a, b)$$

Where m and n are the Bézout coefficients

When the greatest common divisor of two integers is one, the two integers "a" and "b" are said to be co-prime.

Bézout Identity (a,b) are coprime

$$am + bn = 1$$

Where m and n are the Bézout coefficients

When this is holds, the Bézout coefficient "m" is the modular inverse of *(a)mod(b)*; and from the definition of the modular inverse:

Modular Inverse

$$am = 1mod(b)$$

Where m is the modular inverse of (a)mod(b)

Similarly in the co-prime case, the Bézout coefficient "n" is the modular inverse of *(b)mod(a).* Notably, this is an inverse operation as required for asymmetric encryption.

RSA: Key Generation

As previously stated, chose two large prime numbers "p" and "q" and compute their product "N" and the totient ϕ(N); where the totient is the number of positive integers less than N and coprime to N. Notably, If X is a prime number, ϕ(X) = X -1. As p and q are prime, ϕ(N) = (p-1)(q-1). A randomly selected positive integer "E" which is coprime to ϕ(N) and lies within the range 1 < E < ϕ(N) is then chosen. The public key is then:

RSA Public Key

$$(E, N)$$

Where E is Coprime to totient φ(N), 1 < E < φ(N) and N is the product of the two large prime numbers

The EEA is then used to calculate the modular inverse "D" of *(E)modN). D is then used in the Private Key.*

$$DE + K\varphi(\mathrm{N}) = 1 = \gcd(E, \varphi(\mathrm{N}))$$

Where D is the modular inverse of (E)modN). The K Bézout coefficient is relatively insignificant, but should also remain undisclosed.

RSA Private Key

$$(D, N)$$

Where D is the modular inverse of (E)mod(N) and N is the product of the two large prime numbers

RSA: Encryption and Decryption

For encryption, the sender transforms their message "M" into number using a reversible protocol, and using the RSA encryption scheme, obtains the cipher text "C". For this investigation the ASCII scheme was used.

RSA Encryption Scheme

$$C \equiv M^E (mod\ N)$$

Where C is the cipher text, M the numerical message, and E and N from the Public Key

The Decryption is then the inverse operation, via using the Private Key. It is not proven or stated here, but this relies on Fermat's Little Theorem (FLT).

RSA Decryption Scheme

$$M \equiv C^D (mod\ N)$$

Where C is the cipher text, M the numerical message, and E and N from the Public Key

Practically, the numbers used in the RSA modular exponentiation calculations are often too large for direct method computational calculations, and hence more efficient methods are required. The method used in this investigation is the "Right-to-left binary method" which uses the "exponentiation by squaring / square and multiply/ binary exponentiation" principle. This algorithm for computing the RSA encryption and decryption is implemented in the RSAde.m MATLAB programme.

Factorisation Algorithm

As stated, RSA encryption is based on the difficulty of factorising prime numbers, as if N could be easily factorised into p and q then D would be trivial to calculate, and thus cracking the encryption. Currently the common minimum key length N for RSA encryption is 1024 bits, however, the US National Institute of Standards and technology (NIST) advises and RSA key length of 2048 bits. As currently, there is no efficient integer factorisation methodology to factorise key length of this size. For this report, the largest N key used is 165073. Using the Bit length equation, this relates to a bit size of 18. There are multiple factorisation methods for key-lengths of this size such as: "Difference of squares", "Quadratic residues", "Elliptic curve factorisation", and "Quadratic sieve" etc. For this investigation the classical Shor's algorithm was used. Notably, when Quantum computing technology sufficiently matures, prime factorisation will no longer be a difficult enough problem to solve, negating the one-way properties, and RSA encryption will no longer be as secure.

Bit Length

$$BL(N) = \lceil \log_2(N+1) \rceil$$

Where BL is the bit length, N is the RSA key

Overview Shor's Classical Factorisation Algorithm

In order to factorise N, and integer "y" is selected such that it is coprime with N. once a suitable y value is found, it's modular period "r" is calculated. This is the minimum number of steps required for successive powers of y(mod N) until the calculation returns y, as detailed in the modular period equation 1.

Modular period equation 1

$$y^r \equiv 1(mod\ N)$$

Where y is coprime to the RSA key N, and r is the modular period, the first integer to first satisfy the equality

If the modular period for that y value is not an even integer, repeat the process for a new y value. Once an even modular period is calculated, a new value "x" is computed as in modular period equation 2.

Modular period equation 2

$$x \equiv y^{r(0.5)}(mod\ N)$$

Where y is coprime to the RSA key N, r is an even modular period

The two prime factors of N are then calculated as below, using the EEA algorithm:

Factor 1

$$f_1 = \gcd(x+1, N)$$

Factor 2

$$f_2 = \gcd(x-1, N)$$

Such that

$$(f_2)(f_1) = N$$

Where N is the RSA Key, and x is calculated in Modular period equation 2, and the gcd is found using the EEA.

Investigation

Section 1 – ASCII

In this project, the string "Hey Johnny!" was required to be transformed into the ASCII decimal code. As this is a simple conversion using an ASCII chart, this was trivial to conduct in MATLAB using the inbuilt conversion functions "double()" and "char()" to convert to and from ASCII respectively.

Hey Johnny! = 72, 101, 121, 32, 74, 111, 104, 110, 110, 121

An additional procedure was conducted at this point, in order to apply symmetric encryption (shared secret key). First the ASCII form of the string into Binary using the DectoBin.m MATLAB file. A secret

key "S" was then randomly generated from the valid ASCII decimal range of [0 to 127] and also converted to binary. Note, it is the random generation of S (or pseudo random due to the MATLAB computation function randi([0 127])) that secures the message. This method is cracked considerably easier than RSA, without considering the sharing of the private key. The message is then encoded as:

Symmetric Encryption

$$E = (M + S)(mod\ 2)$$

Where E is the encrypted message, M and S are the Message and Key in Binary notation.

Symmetric Decryption1

$$M = (E + S)(mod\ 2)$$

Where E is the encrypted message, M and S are the Message and Key in Binary notation.

This was implemented in the "Hey Johnny.m" MATLAB file, and correctly deciphers the String.

Section 2 – Basic RSA

The string "Hey Johnny!" was then encrypted using the RSA procedure previously described, using the RSA.m MATLAB file. The task required the relevant RSA values to be: prime numbers p = 37 and q = 59, and the public key E = 19. The resultant encoded message is given below.

"Hey Johnny!" Encoded in ASCII Via RSA

372 1470 1823 1337 1739 814 363 147 147 1823 810

Where p = 37, q = 59, Public key (E = 19,N = 2183) and Private Key (D = -989, N = 2183)

Notably the Private Key D is negative in this scenario. As D is the modular inverse, it must satisfy the definition so that DE = 1(mod ϕ(N)). From modular arithmetic's, all mod(N) will have the same value if a multiple of N is added. Therefore, D has ϕ(N) added until it is no longer negative. This new D will still satisfy the modular inverse. Once D was no longer negative, the RSA decryption scheme utilizing the binary exponentiation algorithm (found in RSAde.m MATLAB file) was applied using the RSA.m Programme.

An alternative, but less efficient and potentially erroneous decoding method for dealing with a negative private key D is instead to find the modular inverse of the encryption.

Alternative Decode Method

$$M \equiv C^D(mod\ N) = C^{(-1)-D}(mod\ N)$$

Where D is the Negative Private Key, and C^{-1} is the modular inverse of C(mod N)

If N is not a prime number however, the modular inverse of C may not exist; therefore, potentially resulting in erroneous decoding.

Method 1 RSA decoding "Hey Johnny!" Encoded in ASCII

Hey Johnny!

Method 2 RSA decoding "Hey Johnny!" Encoded in ASCII

Hey 4ɪhnny!

Clearly, from the second decoding, the 5th and 6th characters in the encoding message (1739, 814) have no modular inverse and are thus decoded in correctly.

Section 3 – Further RSA

For this Task investigated, a previously RSA encrypted file in the ASCII decimal cypher with public key (e=307, N = 165073) was required to be cracked and deciphered. For this, N was required to be prime factorised. Using the Shor's Algorithm implemented in "factorprog.m" MATLAB file, the prime factors of 165073 were found to be p = 383 and q = 431. The private key D was then trivially obtained using the EEA in RSAde.m file. D was calculated to be 21937.

The RSA encrypted file (RSA-1a-encrypted.txt) was then imported to be decoded in the same manor as in *Section 2.* Notably, even though N = 165073 which is not a prime number, every Code word had a corresponding modular inverse so that both decoding methods produced the exact same message.

Decoded RSA-1a-encrypted.txt RSA ASCII decimal cypher.

"My greatest concern was what to call it. I thought of calling it 'information,' but the word was overly used, so I decided to call it 'uncertainty.' When I discussed it with John von Neumann, he had a better idea. Von Neumann told me, 'You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, no one really knows what entropy really is, so in a debate you will always have the advantage.' [Claude Shannon explaining why he named his uncertainty function 'entropy', from Scientific American (1971), volume 225, page 180]"

Where Public Key = (307,165073), Private Key = (21937,165073), p = 383, q = 431

Due to the legibility, and the significant relevance of this famous quote, I am significantly certain that is the correctly deciphered message.

Digital Signature Algorithm (DSA) Encryption

Digital Signatures: Overview

Digital signatures, similar to their classic signature counter parts, are used to authenticate messages. For Digital signature schemes to be valid, they must authenticate that the message was indeed sent by the intended party, be unfalsifiable, non-reusable and irrevocable. All of these are intrinsically valuable properties in security. As mentioned earlier, the RSA can be used to develop digital signatures In a very similar process, only this time instead of only the receiver being able to decrypt the message, and everyone able to use the public key to send an encrypted message; the properties are reversed. The owner of the Public key (the original receiver previously) encrypts their message using their Private Key, and everyone may decode it using the public key.

RSA Digital Signature

$$S \equiv M^D (mod\ N)$$

Where S is the signed message, M the numerical message, D the private key and N from the Public Key

RSA Digital Signature Verification (Receiver)

$$M \equiv S^E (mod\ N)$$

Where S is the signed message, M the numerical message, and E and N from the Public Key

Intuitively, a different Public key and Private Key should be used than that for the previous encryption process, and it is recommended to use different keys for each message.

Notably this process essentially encrypts the entire message, when only a "signature" is required. This is inefficient. Instead, Hash functions are used to reduce the long message into a shorter fixed bit length string which are then signed instead.. These Hash functions must also be One-way functions and be Collision resistant. Rudi mentally, this means that any two messages feasibly cannot have the same signature profile, or at least impractical to replicate. Otherwise forgeries are possible.

DSA

DSA is an alternative method for generating digital signatures, and is based on the intractable discrete logarithm problem, as opposed to RSA's Numerical factorisation. Similar to RSA, Hash functions are used and messages are signed using the private key.

DSA Key generation

Choose two key bit- lengths "L" and "N", where N is less than or equal to the length of the hash output. A typical pairing is (L = 2048, N = 224). Choose a prime number "q" of bit-length N, and a prime "p" such that p -1 is a multiple of q, and p is bit-length L. A new number g is calculated such that its multiplicative order modulo p is q.

Multiplicative order

$$g^q \equiv 1 (mod\ p)$$

Where q is the smallest integer which satisfies the equality, and g and p are coprime. This is inherently related to the Discrete Logarithm.

This is calculated via:

$$g = h^{\frac{p-1}{q}} mod(p)$$

Where h is arbitrary (1<h<p-1) which satisfies the equality. Shor's algorithm may be used.

A randomly chosen integer "a" in the range (1<a<p-1). This is the signer's private key and is used to generate:

$$A = g^a (mod\ p)$$

The Signer's public key is then:

DSA Public Key

$$(p, q, g, A)$$

DSA Signature Generation

A public hash function *H* is chosen for the message "m", and another random number "K" in the range 1<K<q generated. This is the used to calculate:

$$r = (g^K (mod\ p))(mod\ q)$$

So long as r>0

The modulo inverse (K^{-1}) of K(mod q) is calculated using the EEA and then used to calculate:

$$s = K^{-1}(H(m) + ar)(mod\ q)$$

And the signature is then:

DSA Signature of message m

$$(r, s)$$

DSA Signature Verification

In order to verify the signature (r,s) for message "m", the verifier uses the public key to check that:

$$0 < r < q\ and\ 0 < s < q$$

If this holds then the following process is conducted:

$$w = S^{-1}(mod\ q)$$
$$y = (H(m)(w))(mod\ q)$$
$$z = (rw)(mod\ q)$$
$$V = \big((g^y A^z)(mod\ p)\big) mod q$$

Where all symbols take their previous meanings, and the new symbols are for ease of reading

If the signature is valid, then:

$$V = r$$

This is due to:

$$g^y A^z \equiv g^{s^{-1}(H(m)+ra)} \equiv g^K (mod p)$$

Substituting in to V:

$$V = (g^K (mod p))(mod q) \equiv r$$

Which is the same form as the definition of r

The Documentation with the full proof of this is provided in the appendix.

Circumventing DSA and Applications

As stated previously, the security of DSA relies on the intractability of the discrete logarithm problem. An alternative approach is to pray on the pseudo-random number generator used in the calculations. If the generator does not produce a sufficiently apparent stochastic scheme, then the next random number may be predicted from the pattern.
One approach to solve the discrete logarithm problem applies the Chinese remainder theorem and uses "Shank's Baby-step giant step (BSGS) algorithm" (note: BSGS may also be used to solve this problem). This method is known as the Pohlig Hellman algorithm, and sources for more information are available in the appendix.

A variation of DSA, known as Elliptic Curve Digital Signature Algorithm (ECDSA) has become more popular in recently, and It is now the recommended process for digital signing by NIST. ECDSA uses the algebraic structure of elliptic curves; such as point addition and point doubling. These properties are used in the generation of the public key (from a randomly selected private key) in order to generate the signature, with further information sources available in the appendix.

One specific use case for ECDSA is in the security of the "Block-chain" and Bitcoin. ECDSA is used to generate the digital signatures for transactions, along with 256 bit encryption Hash functions (eg SHA256). The digital signatures are used to verify transactions on the digital transaction "ledger". And recall from DSA that the digital signatures are unique for each transactions due to the Hash functions. The DSA signatures verify to whom the transactions debit and credit.

As an aside, the security of the Block chain is further based upon Hash functions. The validity of the ledger of transactions is split into transaction "blocks". All the information enclosed in this block of transactions is passed through the hash functions, producing a 256bit value. However, for a valid block of transactions, the hash value must start with 30 zeros. Probabilistically, this would occur randomly as 2^{-30}. Due to the one-way properties of the Hash function, this is only feasibly done computationally "brute forcing" to find an otherwise arbitrary value when added to the block which makes the resultant Hash value has this property, which is intuitively tremendous computational work. The valid transaction ledger is then chosen as the Block-chain which is the longest, and therefore has the most computational work. As multiple computers are to solve for this Hash function at the same time (in order to obtain cryptocurrencies), a computer with a fraudulent ledger would have to constantly solve the hash function before every other computer, which is what probabilistically ensures the validity and security of the block-chain.

Conclusion

The RSA scheme was able to correctly encrypt and decipher the tasked messages; however, the Shor's factorisation algorithm was able to crack the subsequent RSA encryption. With further advancements in computational power, it is likely that this will be sufficient to solve prime factorisation for larger bit-length encryption keys, especially if quantum computing technology matures. Digital signatures are an extremely useful tool, but with the constant advancements in computational power, the process behind them must also become more complex or computational intensive in order to remain useful.

Bibliography

LinkÓping University DSA Cryptology Lectures - https://www.icg.isy.liu.se/courses/tsit03/forelasningar/cryptolecture08.pdf

Johannes Buchman, 2001, *The Digital Signature - Algorithm*https://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1003_DSA.pdf

Kefa Rabah , 2005. *Security of the Cryptographic Protocols Based on Discrete Logarithm Problem. Journal of Applied Sciences, 5: 1692-1712.*

Eric Rykwalder, 2014, *The Math Behind Bitcoin, Coindesk* https://www.coindesk.com/math-behind-bitcoin

Eray Altili, 2018, *Cryptology, Encryption, Hash Functions and Digital Signature, Medium Corporation* https://medium.com/@ealtili/cryptography-encryption-hash-functions-and-digital-signature-101-298a03eb9462

Prof Bill Buchanan OBE, 2018, *Integer Factorisation – Defining the Limits of RSA Cracking, Medium Corporation* – https://medium.com/coinmonks/integer-factorization-defining-the-limits-of-rsa-cracking-71fc0675bc0e

Brian Curran, 2018, *What is RSA Cryptography?, Blockonomi.com* https://blockonomi.com/rsa-cryptography/#

2015, *The Beauty of Asymmetric Encryption*, Dusted Codes – https://dusted.codes/the-beauty-of-asymmetric-encryption-rsa-crash-course-for-developers

Word Count: 3206

Appendix

Mathematical Proof of DSA:

William Stallings, Supplement to *Cryptography and Network Security, Fifth Edition* – http://mercury.webster.edu/aleshunas/COSC%205130/K-DSA.pdf

Solving the Discrete Logarithm problem:

https://scialert.net/fulltext/?doi=jas.2005.1692.1712

Pohlig – Hellman Algorithim:

https://scialert.net/fulltext/?doi=jas.2005.1692.1712

https://medium.com/@__cpg/solving-the-discrete-log-problem-using-the-chinese-remainder-theorem-56a369bdcace

https://www.cryptologie.net/article/196/pohlig-hellman-algorithm/

ECDSA & Bitcoin

Eric Rykwalder, 2014, *The Math Behind Bitcoin, Coindesk*
https://www.coindesk.com/math-behind-bitcoin